DCOP Games for Multi-agent Coordination

Jonathan P. Pearce, Rajiv T. Maheswaran and Milind Tambe

University of Southern California, Los Angeles, CA 90089, USA
{jppearce, maheswar, tambe}@usc.edu

Abstract. Many challenges in multi-agent coordination can be modeled as distributed constraint optimization problems (DCOPs) but complete algorithms do not scale well nor respond effectively to dynamic or anytime environments. We introduce a transformation of DCOPs into graphical games that allows us to devise and analyze algorithms based on local utility and prove the monotonicity property of a class of such algorithms. The game-theoretic framework also enables us to characterize new equilibrium sets corresponding to a given degree of agent coordination. A key result in this paper is the discovery of a novel mapping between finite games and coding theory from which we can determine *a priori* bounds on the number of equilibria in these sets, which is useful in choosing the appropriate level of coordination given the communication cost of an algorithm.

1 Introduction

A distributed constraint optimization problem (DCOP) [9, 11] is a useful formalism in settings where distributed agents, each with control of some variables, attempt to optimize a global objective function characterized as the aggregation of distributed constraint utility functions. DCOPs can be applied for coordination in multi-agent domains, including sensor nets, distributed spacecraft, disaster rescue simulations, and software personal assistant agents. For example, sensor agents may need to choose appropriate scanning regions to optimize targets tracked over the entire network, or personal assistant agents may need to schedule multiple meetings in order to maximize the value of their users' time. As the scale of these domains become large, current complete algorithms incur immense computation costs. A large-scale network of personal assistant agents would require global optimization over hundreds of agents and thousands of variables, which is currently very expensive. Though heuristics that significantly speed up convergence have been developed [8], the complexity is still prohibitive in large-scale domains. On the other hand, if we let each agent or variable react on the basis of its local knowledge of neighbors and constraint utilities, we create a system that scales up very easily and is far more robust to dynamic environments.

Recognizing the importance of local search algorithms, researchers initially introduced DBA[12] and DSA[1] for Distributed CSPs, which were later extended to DCOPs with weighted constraints [13]. While detailed experimental analyses of these algorithms on DCOPs is available[13], we still lack theoretical tools that allow us to understand the evolution and performance of such algorithms on arbitrary DCOPs. To provide such tools, this paper decomposes a DCOP into an equivalent graphical *DCOP game*, which differs from graphical games with general reward functions [4, 10]. DCOP

games not only allow us to analyze existing local search algorithms, they also suggest an evolution to *k-coordinated* algorithms, where a collection of k agents coordinate their actions in a single negotiation round, which leads to new notions of equilibria. For example, a 2-coordinated algorithm would be an algorithm in which at most two agents could coordinate their actions, and a 2-coordinated equilibrium would be a situation in which no 2-coordinated algorithm could improve the quality of the assignment of values to variables. A key contribution of this paper is the application of a mapping between finite games and coding theory to determine *a priori* bounds on cardinality of equilibria sets of k-coordinated algorithms. Such bounds could be used to help determine an appropriate level of coordination for agents to use to reach an assignment of variables to values, in situations where the cost of coordination between multiple agents must be weighed against the quality of the solution reached.

2 DCOP Games, *k*-Coordinated Equilibria Sets and Bounds

We begin with a formal representation of a distributed constraint optimization problem and an exposition to our notational structure. Let $V \equiv \{v_i\}_{i=1}^{N}$ denote a set of variables, each of which can take a value $v_i = x_i \in X_i$, $i \in \mathcal{N} \equiv \{1, \ldots N\}$. Here, X_i will be a domain of finite cardinality $\forall i \in \mathcal{N}$. Interpreting each variable as a node in a graph, let the symmetric matrix E characterize a set of edges between variables/nodes such that $E_{ij} = E_{ji} = 1$ if an edge exists between v_i and v_j and $E_{ij} = E_{ji} = 0$, otherwise ($E_{ii} = 0\ \forall i$). For each pair (i, j) such that $E_{ij} = 1$, let $U_{ij}(x_i, x_j) = U_{ji}(x_j, x_i)$ represent a reward obtained when $v_i = x_i$ and $v_j = x_j$. We can interpret this as a utility generated on the edge between v_i and v_j, contingent simultaneously on the values of both variables and hence referred to as a *constraint*. The global or team utility $\overline{U}(x)$ is the sum of the rewards on all the edges when the variables choose values according to the assignment $x \in X \equiv X_1 \times \cdots \times X_N$. Thus, the goal is to choose an assignment, $x^* \in X$, of values to variables such that

$$x^* \in \arg\max_{x \in X} \overline{U}(x) = \arg\max_{x \in X} \sum_{i,j:E_{ij}=1} U_{ij}(x_i, x_j)$$

where x_i is the i-th variable's value under an assignment vector $x \in X$. This constraint *optimization* problem completely characterized by (X, E, U), where U is the collection of constraint utility functions, becomes *distributed* in nature when control of the variables is partitioned among a set of autonomous agents. For the rest of this paper, we make the simplifying assumption that there are N agents, each in control of a single variable.

We present a decomposition of the DCOP into a game as follows. Let v_j be called a *neighbor* of v_i if $E_{ij} = 1$ and let $\mathcal{N}_i \equiv \{j : j \in \mathcal{N}, E_{ij} = 1\}$ be the indices of all neighbors of the i-th variable. Let us define $x_{-i} \equiv [x_{j_1} \cdots x_{j_{K_i}}]$, hereby referred to as a *context*, be a tuple which captures the values assigned to the $K_i \equiv |\mathcal{N}_i|$ neighboring variables of the i-th variable, i.e. $v_{j_k} = x_{j_k}$ where $\cup_{k=1}^{K_i} j_k = \mathcal{N}_i$.

In a DCOP game, for an assignment x, we define a utility function $u_T(x)$ for a team of agents, $T \subseteq \mathcal{N}$ to be the sum of the utilities on all constraint links for which at least

one vertex represents an agent in the team, i.e.

$$u_T(x) = \sum_{i \in T} \sum_{j: E_{ij}=1} U_{ij}(x_i, x_j) - \sum_{i \in T} \sum_{j \in T, j>i, E_{ij}=1} U_{ij}(x_i, x_j).$$

The utility for a single agent ($T = \{i\}$) is

$$u_i(x) \equiv \sum_{j \in \mathcal{N}_i} U_{ij}(x_i, x_j)$$

Thus, in a DCOP game, team utilities are not the sums of individual utilities. We now have a *DCOP game* defined by (X, E, u_T) where u_T is a collection of the utility functions for all teams.

In current local algorithms, agents change values based on anticipated payoffs of only their own utilities. Since DCOPs are inherently cooperative, it is natural for agents to coordinate in order to improve global solution quality. DCOP games provide a framework to analyze, categorize and evaluate such multi-agent coordination. Let us define a *k-concurrent deviation* from an assignment x to be an assignment $\tilde{x}$ where exactly k of the N variables (agents) have values different from x, i.e. $d(x, \tilde{x}) \equiv |\{i : x_i \neq \tilde{x}_i\}| = k$. We now introduce the notion of a *k-coordinated equilibrium*, defined to be an assignment x^* such that if $\hat{k} \leq k$,any $\hat{k}$-concurrent deviation $\tilde{x}$ from x^*, i.e. $d(x^*, \tilde{x}) \leq \hat{k}$, cannot improve the team utility for the set of agents which deviated, $D(x^*, \tilde{x}) \equiv \{i : x_i^* \neq \tilde{x}_i\} \subseteq \mathcal{N}$. A 1-coordinated equilibrium is identical to a Nash equilibrium as $|D(x^*, \tilde{x})| = d(x^*, \tilde{x}) = 1$ is a unilateral deviation and the team utility u_T reduces to the utility u_i for a single agent. Let $X_{kE} \subseteq X$ be the subset of the assignment space which captures all k-coordinated equilibrium assignments:

$$X_{kE} \equiv \{x \in X : \tilde{x} \in X, 1 \leq d(x, \tilde{x}) \leq k \Rightarrow u_{D(x,\tilde{x})}(x) \geq u_{D(x,\tilde{x})}(\tilde{x})\}.$$

Proposition 1. *If x^* optimizes a DCOP characterized by (X, E, U), then $x^* \in X_{kE}$ $\forall$ $k \in \mathcal{N}$.*

Proof. Let us assume that x^* optimizes the DCOP (X, E, U) and $x^* \notin X_{kE}$ for some $k \in \mathcal{N}$. Then, there exists some $\tilde{x} \in X$ such that $u_{D(x^*,\tilde{x})}(x^*) < u_{D(x^*,\tilde{x})}(\tilde{x})$. By adding

$$\sum_{i \notin D(x^*,\tilde{x})} \sum_{j \notin D(x^*,\tilde{x}), j>i} U_{ij}(x_i^*, x_j^*) = \sum_{i \notin D(x^*,\tilde{x})} \sum_{j \notin D(x^*,\tilde{x}), j>i} U_{ij}(\tilde{x}_i, \tilde{x}_j)$$

to both sides, we can show $\overline{U}(x^*) < \overline{U}(\tilde{x})$, which is a contradiction. ■

Simply put, the proposition states that the optimal solution to the DCOP is a k-coordinated equilibrium for all k up to the number of variables in the system. In our DCOP framework, we are optimizing over a finite set. Thus, we are guaranteed to have an assignment that yields a maximum. By the previous proposition, this assignment is an element of X_{kE} $\forall k \in \mathcal{N}$, including $k = 1$. Thus, we are guaranteed the existence of a pure-strategy Nash equilibrium. This claim cannot be made for any arbitrary graphical game [4, 10]. Furthermore, from the definition above we see that for $k = 1, \ldots, N-1$, we have $X_{(k+1)E} \subseteq X_{kE}$ because if $x \in X_{(k+1)E}$, we have

$$d(x, \tilde{x}) \leq k + 1 \Rightarrow u_{D(x,\tilde{x})}(x) \geq u_{D(x,\tilde{x})}(\tilde{x})$$

which implies $d(x, \tilde{x}) \leq k \Rightarrow u_{D(x,\tilde{x})}(x) \geq u_{D(x,\tilde{x})}(\tilde{x})$ and thus, $x \in X_{kE}$. Thus, as k increases, the sets of k-coordinated equilibria can be pictured as a series of smaller and smaller concentric circles, culminating in a single point, representing the k-coordinated equilibrium for $k = N$, which is also the optimal solution to the DCOP.

In our notation X_{1E} characterizes the set of all Nash equilibria (no unilateral deviations) and X_{NE} characterizes the set of assignments that maximize global utility (no N-agent deviations).

We exploit the sets X_{kE} in the design of a new class of DCOP local algorithms, and analysis of their equilibrium points. In particular, for a given algorithm α, let Z_α denote the set of assignments at which the algorithm will remain stationary, i.e. the terminal states. An algorithm α is *k-coordinated* if $Z_\alpha \subseteq X_{kE}$ and $Z_\alpha \not\subseteq X_{(k+1)E}$ for $k < N$ or $Z_\alpha \subseteq X_{NE}$ for $k = N$.

Example 1. **Meeting Scheduling.** Consider two agents trying to schedule a meeting at either 7:00 AM or 1:00 PM with the constraint utility as follows: $U(7, 7) = 1, U(7, 1) = U(1, 7) = -100, U(1, 1) = 10$. If the agents started at $(7, 7)$, any 1-coordinated algorithm would not be able to reach the global optimum, while 2-coordinated algorithms would.

Section 3 illustrates that existing local DCOP algorithms are special cases of such k-coordinated algorithms with $k = 1$, and $k \geq 2$ may improve solution quality but at a higher communication cost.

Choosing an appropriate level of k-coordination given the higher communication cost is thus a critical question, similar to the choice of neighborhood size in large-neighborhood search in centralized constraint satisfaction. We assume that k-coordinated algorithms are capable of searching any neighborhood of size k completely, although the price for this completeness must be paid in the increasing number of messages required to ensure a k-equilibrium for increasing k.

To begin answering this question, we provide *a priori* bounds on the number of equilibria in sets X_{kE}, e.g. a significant reduction in number of equilibria may justify a jump from k-coordination to $(k + 1)$ coordination.

We first consider games, where each player (agent) can choose among q strategies (values), i.e. $|X_i| = q, \ \forall i \in \mathcal{N}$. We assume that the payoff structure is such that the optimal k-concurrent response to any context of cardinality $N - k$ is unique. Otherwise, any bound can be violated in the case where all assignments yield identical utilities and every assignment is an optimal equilibrium point. Furthermore, we assume that agents have the ability to communicate with all other agents to facilitate all k-concurrent deviations (although such communication may be indirect, requiring message relay).

To find upper bounds for the number of k-coordinated equilibria in such games, we discovered a correspondence from games to coding theory [6, 5]. A fundamental problem in the theory of error-correcting codes is the determination of appropriate codewords to use in a code. The code designer must balance the need for brevity, expressiveness, and error-correctability of the code, determined, respectively, by the length, maximum number, and distinctiveness of the allowed codewords. A common measure of the distinctiveness of two codewords is the Hamming distance, which is defined as the number of places at which the codewords differ.

For our purposes, an assignment is analogous to a codeword of length N from an alphabet of cardinality q (Each variable in the DCOP maps to a place in a codeword,

and each member of the domain of the variables maps to a member of the alphabet from which the codewords are created). An assignment $\tilde{x}$ which is a k-concurrent deviation from an assignment x, can also be interpreted as two codewords with a Hamming distance of k, where $d(x, \tilde{x}) \equiv |\{i : x_i \neq \tilde{x}_i\}| = k$ as stated earlier. If x_1 is a k-coordinated equilibrium and $\tilde{x}_1$ is a k-concurrent deviation from x_1, $\tilde{x}_1$ cannot be a k-coordinated equilibrium point because $u_{D(x,\tilde{x})}(x) > u_{D(x,\tilde{x})}(\tilde{x})$ since there is a unique optimal response to the context $\{x_i : i \in \mathcal{N} \setminus D(x, \tilde{x})$. Thus, if x_2 is a different k-coordinated equilibrium, then x_2 cannot be reachable from x_1 via a k-concurrent deviation (and vice-versa). In the language of coding theory, x_1 and x_2 must be separated by a Hamming distance greater than k. The problem of finding the maximum possible number of k-coordinated equilibria can then be reduced to finding the maximum number of codewords in a codespace of size q^N such that the the minimum distance among any two codewords is $d = k + 1$.

In coding theory literature, a q-ary (n, M, d) code refers to a collection of length n words constructed over an alphabet A of cardinality q where M codewords are chosen such that the minimum Hamming distance between any two codewords is at least d. Let $A_q(n, d) \equiv \max\{M : \exists \text{ an } (n, M, d) \text{ code over alphabet } A\}$. Three well-known bounds for $A_q(n, d)$ are the Hamming bound:

$$A_q(n, d) \leq q^n / \left(\sum_{i=0}^{\lfloor (d-1)/2 \rfloor} \binom{n}{i} (q-1)^i \right)$$

the Singleton bound:

$$A_q(n, d) \leq q^{n-d+1}$$

and the Plotkin bound:

$$A_q(n, d) \leq \left\lfloor \frac{d}{d - rn} \right\rfloor$$

Note that the Plotkin bound is only valid when $rn < d$, where $r = 1 - q^{-1}$, and $A_q^S(n, d) = q^{n-d+1}$ [5]. For the special case of binary ($q = 2$) codes, we can use the relation

$$A_q(n, 2r - 1) = A_q(n_1, 2r)$$

[6] to obtain tighter bounds for even distances using the Hamming bounds for odd distances. Thus, the number of k-coordinated equilibria for a given n, q and $d = k + 1$ can be bounded by the tightest of the bounds mentioned above.

For non-binary codes, we note that the Hamming bound is identical for d and $d + 1$ when d is odd. The Hamming bound is derived by using a sphere packing argument that states that the number of words q^n must be greater than the number of codewords $A_q(n, d)$ times the size of a sphere centered around each codeword. A sphere $S_A(u, r)$ with center u and radius r is the set $\{v \in A^n : d(u, v) \leq r\}$. It can be shown that $S_A(u, r)$ in A^n contains exactly $\sum_{i=0}^{r} \binom{n}{i} (q-1)^i$ words. If d is odd, the tightest packing then occurs with spheres of radius $(d-1)/2$ and each word can be uniquely assigned to the sphere of a codeword closest to it. If d is even, it is possible for a word to be equidistant from two codewords and it is unclear how to assign this word to a sphere. The Hamming bound

addresses this issue by simply using the bound obtained with the smaller distance $d-1$, which leads to smaller spheres and hence a larger bound than necessary. In essence, this ignores the contribution of a word that lies on the "boundary" to the volume of a sphere. We show one can appropriately partition these boundary assignments to achieve tighter bounds.

Proposition 2. *For even d,*

$$A_q(n,d) \leq \min\left\{ \frac{q^n - \binom{n}{d/2}(q-1)^{d/2}}{\sum_{i=0}^{\lfloor (d-1)/2 \rfloor}\binom{n}{i}(q-1)^i}, \frac{q^n}{\sum_{i=0}^{\lfloor (d-1)/2 \rfloor}\binom{n}{i}(q-1)^i + \binom{n}{d/2}(q-1)^{d/2}(\frac{1}{n})} \right\}.$$

Proof. It is clear that any word that has Hamming distance $\lfloor (d-1)/2 \rfloor$ or less from a codeword belongs in the sphere of that codeword, because belonging to more than one sphere under those conditions would violate the distance requirement of the code. Given an even distance, each codeword will see $\binom{n}{d/2}(q-1)^{d/2}$ words that are $d/2$ away from it. It cannot claim all those words as other codewords may be seeing the same words. We do know however that each of the words on the boundary can be seen by at most n codewords as a word of length n can be on the boundary of at most n spheres. Furthermore, each word on a boundary can be seen by at most $A_q(n,d)$ codewords, i.e. the number of codewords in the space. Thus, each codeword can safely incorporate $1/\min\{n, A_q(n,d)\}$ of each boundary word into its sphere. Aggregating over all the words on the boundary, we can increase the volume of the sphere by $\binom{n}{d/2}(q-1)^{d/2}/\min\{n, A_q(n,d)\}$. Using the sphere packing argument, if $A_q(n,d) \leq n$, we have

$$q^n \geq A_q(n,d)\left[\sum_{i=0}^{\lfloor (d-1)/2 \rfloor} \binom{n}{i}(q-1)^i + \frac{\binom{n}{d/2}(q-1)^{d/2}}{A_q(n,d)} \right]$$

$$\Rightarrow A_q(n,d) \leq \frac{q^n - \binom{n}{d/2}(q-1)^{d/2}}{\sum_{i=0}^{\lfloor (d-1)/2 \rfloor}\binom{n}{i}(q-1)^i} \equiv G_1,$$

and if $A_q(n,d) \geq n$, we have

$$q^n \geq A_q(n,d)\left[\sum_{i=0}^{\lfloor (d-1)/2 \rfloor} \binom{n}{i}(q-1)^i + \frac{\binom{n}{d/2}(q-1)^{d/2}}{n} \right]$$

$$\Rightarrow A_q(n,d) \leq \frac{q^n}{\sum_{i=0}^{\lfloor (d-1)/2 \rfloor}\binom{n}{i}(q-1)^i + \binom{n}{d/2}(q-1)^{d/2}(\frac{1}{n})} \equiv G_2.$$

Now, we have $A_q(n,d) \leq n \Rightarrow A_q(n,d) \leq G_1$ and $A_q(n,d) \geq n \Rightarrow A_q(n,d) \leq G_2$. We can show that $G_1 \odot n \Leftrightarrow G_2 \odot n,\ \forall \odot \in \{<, >, =\}$. Furthermore, $G_1 \odot n, G_2 \odot n \Leftrightarrow G_1 \odot G_2$. Thus, when $G_1 < n, G_2 < n$, we have both that G_2 is invalid and G_1 is the tighter bound and when $G_1 > n, G_2 > n$, G_1 is invalid and G_2 is the tighter bound. We can then express the bound as

$$A_q(n,d) \leq \min\{G_1, G_2\}.\blacksquare$$

We refer to this as the *modified Hamming bound*. The new bound appears to dominate other bounds for sufficiently large n, for even d and $q > 2$. In Figure 1, we illustrate the usefulness of our new bound.

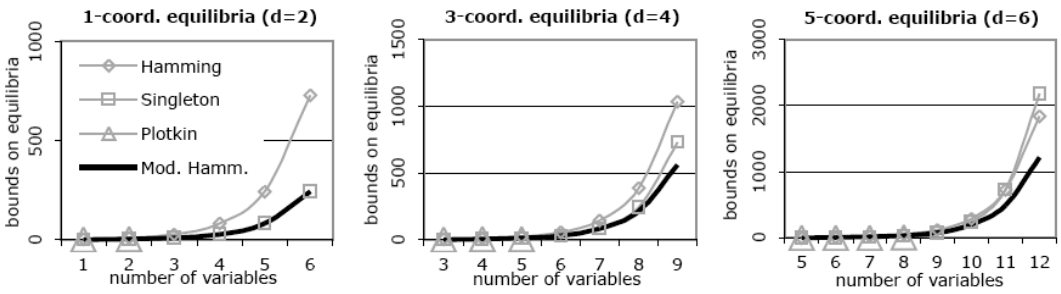

Fig. 1. Modified Hamming Bound

3 DCOP Algorithms: Analysis and Design

The *DCOP game* perspective also aids in the analysis of existing local-utility based algorithms and design of key new algorithms. Among existing DCOP algorithms, the first is the MGM (Maximum Gain Message) algorithm which is a modification of DBA (Distributed Breakout Algorithm) [12] focused solely on gain message passing. DBA cannot be directly applied because there is no global knowledge of solution quality which is necessary to detect local minima. The second is DSA (Distributed Stochastic Algorithm) [1], which is a homogeneous stationary randomized algorithm.

These algorithms work as follows: For synchronous running, let us define a *round* as the duration between a change in assignment for a particular algorithm. A single round could involve multiple broadcasts of *messages*. Every time a messaging phase occurs in a round, we will count that as one *cycle* and cycles will be our performance metric for speed, as is common in DCOP literature. Let $x^{(n)} \in X$ denote the assignments at the beginning of the n-th round. We assume that every algorithm will broadcast its current value to all its neighbors at the beginning of the round taking up one cycle. Once agents are aware of their current contexts, they will go through a process as determined by the specific algorithm to decide which of them will be able to modify their value. For MGM, each agent broadcasts a gain message to all its neighbors that represents the maximum change in its local utility if it is allowed to act under the current context. An agent is then allowed to act if its gain message is larger than all the gain messages it receives from all its neighbors (ties can be broken through variable ordering or other methods). For DSA, each agent generates a random number from a uniform distribution on $[0, 1]$ and acts if that number is less than some threshold p (the agent will only change value if there is a local utility gain). We note that MGM has a cost of two cycles per round while DSA has a cost of only one cycle per round.

Given the game-theoretic perspective introduced earlier, we recognize that MGM and DSA are in effect k-coordinated algorithms, where $k = 1$. In particular, these algorithms allow only unilateral actions by single agents in a given context. One method to improve the solution quality is for agents to coordinate actions with their neighbors, thus giving rise to k-coordinated algorithm classes. We define two such classes as MGM-k and SCA-k (Stochastic Coordination Algorithm), which facilitate monotonic and randomized evolution, respectively. DSA is in the SCA family of algorithms,

namely SCA-1. In these k-coordinated algorithms, teams of up to k agents can coordinate value updates in order to maximize $u_T(x)$ where T is the set of agents in the team.

Instantiating this concept in SCA-2, we allow agents to make offers to neighboring agents to perform a joint change of value, such that the sum of the utilities of the two agents will increase. They become committed partners if the offer receiver determines that team utility yields a greater gain than its unilateral move. To determine the roles of offerer or receiver, each agent generates a random number from a uniform distribution on $[0, 1]$ and becomes an offerer if that number is less than some threshold q, and a receiver otherwise.

Let $M^{(n)} \subseteq \mathcal{N}$ denote the set of agents allowed to modify the values in the n-th round. In SCA-2, $M^{(n)}$ includes all members of committed teams and uncommitted agents who update with probability p. In MGM-2, additional rounds of message exchanges ensures that if $i \in M^{(n)}$, then i belongs to a team (possibly a team of one) whose gain is larger than the gains of the teams of all its neighbors.

MGM, DSA, and MGM-2 are presented in full in the appendix.

Through our game-theoretic framework, we are able to prove the following monotonicity property of MGM-k, where teams of up to k agents can be formed.

Proposition 3. *When applying MGM, the global utility $\overline{U}(x^{(n)})$ is strictly increasing with respect to the round (n) until $x^{(n)} \in X_{NE}$.*

Proof. We assume $M^{(n)} \neq \emptyset$, otherwise we would be at a Nash equilibrium. When utilizing MGM, if $i \in M^{(n)}$ and $E_{ij} = 1$, then $j \notin M^{(n)}$. If the i-th variable is allowed to modify its value in a particular round, then its gain is higher than all its neighbors gains. Consequently, all its neighbors would have received a gain message higher than their own and thus, would not modify their values in that round. Because there exists at least one neighbor for every variable, the set of agents who cannot modify their values is not empty: ${M^{(n)}}^C \neq \emptyset$. We have $x_i^{(n+1)} \neq x_i^{(n)}\ \forall i \in M^{(n)}$ and $x_i^{(n+1)} = x_i^{(n)}\ \forall i \notin M^{(n)}$. Also, $u_i(x_i^{(n+1)}; x_{-i}^{(n)}) > u_i(x_i^{(n)}; x_{-i}^{(n)})\ \forall i \in M^{(n)}$, otherwise the i-th player's gain message would have been zero. Looking at the global utility, we have

$$
\begin{aligned}
&\overline{U}\left(x^{(n+1)}\right) \\
&= \sum_{i,j:E_{ij}=1} U_{ij}\left(x_i^{(n+1)}, x_j^{(n+1)}\right) \\
&= \sum_{\substack{i,j:i\in M^{(n)},\\ j\in M^{(n)},E_{ij}=1}} U_{ij}\left(x_i^{(n+1)}, x_j^{(n+1)}\right) + \sum_{\substack{i,j:i\in M^{(n)},\\ j\notin M^{(n)},E_{ij}=1}} U_{ij}\left(x_i^{(n+1)}, x_j^{(n+1)}\right) \\
&\quad + \sum_{\substack{i,j:i\notin M^{(n)},\\ j\in M^{(n)},E_{ij}=1}} U_{ij}\left(x_i^{(n+1)}, x_j^{(n+1)}\right) + \sum_{\substack{i,j:i\notin M^{(n)},\\ j\notin M^{(n)},E_{ij}=1}} U_{ij}\left(x_i^{(n+1)}, x_j^{(n+1)}\right) \\
&= \sum_{\substack{i,j:i\in M^{(n)},\\ j\notin M^{(n)},E_{ij}=1}} U_{ij}\left(x_i^{(n+1)}, x_j^{(n)}\right) + \sum_{\substack{i,j:i\notin M^{(n)},\\ j\in M^{(n)},E_{ij}=1}} U_{ij}\left(x_i^{(n)}, x_j^{(n+1)}\right) + \sum_{\substack{i,j:i\notin M^{(n)},\\ j\notin M^{(n)},E_{ij}=1}} U_{ij}\left(x_i^{(n)}, x_j^{(n)}\right)
\end{aligned}
$$

$$= \sum_{i \in M^{(n)}} u_i\left(x_i^{(n+1)}; x_{-i}^{(n)}\right) + \sum_{j \in M^{(n)}} u_j\left(x_j^{(n+1)}; x_{-j}^{(n)}\right) + \sum_{\substack{i,j: i \notin M^{(n)}, \\ j \notin M^{(n)}, E_{ij}=1}} U_{ij}\left(x_i^{(n)}, x_j^{(n)}\right)$$

$$> \sum_{i \in M^{(n)}} u_i\left(x_i^{(n)}; x_{-i}^{(n)}\right) + \sum_{j \in M^{(n)}} u_j\left(x_j^{(n)}; x_{-j}^{(n)}\right) + \sum_{\substack{i,j: i \notin M^{(n)}, \\ j \notin M^{(n)}, E_{ij}=1}} U_{ij}\left(x_i^{(n)}, x_j^{(n)}\right)$$

$$= \sum_{\substack{i,j: i \in M^{(n)}, \\ j \notin M^{(n)}, E_{ij}=1}} U_{ij}\left(x_i^{(n)}, x_j^{(n)}\right) + \sum_{\substack{i,j: i \notin M^{(n)}, \\ j \in M^{(n)}, E_{ij}=1}} U_{ij}\left(x_i^{(n)}, x_j^{(n)}\right) + \sum_{\substack{i,j: i \notin M^{(n)}, \\ j \notin M^{(n)}, E_{ij}=1}} U_{ij}\left(x_i^{(n)}, x_j^{(n)}\right)$$

$$= \overline{U}\left(x^{(n)}\right).$$

The second equality is due to a partition of the summation indexes. The third equality utilizes the properties that there are no neighbors in $M^{(n)}$ and that the values for variables corresponding to indexes not in $M^{(n)}$ in the $(n + 1)$-th round are identical to the values in the n-th round. The strict inequality occurs because agents in $M^{(n)}$ must be making local utility gains. The remaining equalities are true by definition. Thus, MGM yields monotonically increasing global utility until equilibrium. ■

Furthermore, it is clear that an equilibrium will be reached because this algorithm can be mapped to a discrete Hopfield model in which agents act as neurons which ”fire” by choosing a value. It has been shown that such networks always reach local equilibrium [3].

But why is monotonicity important? In anytime domains where communication may be halted arbitrarily and existing strategies must be executed, randomized algorithms risk being terminated at highly undesirable assignments. Given a starting condition with a minimum acceptable global utility, monotonic algorithms guarantee lower bounds on performance in anytime environments.

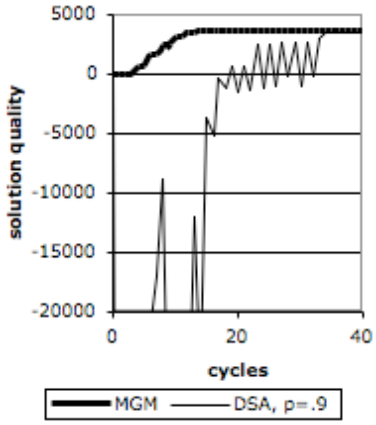

Fig. 2. MGM and DSA for a High-Stakes Scenario

Consider the example in Figure 2 which displays a sample trajectory for both MGM and DSA with identical starting conditions for a high-stakes scenario with 40 variables with three values each. Here, if two neighboring agents take the same value, a penalty of -1000 is incurred. If they take different values, they obtain a reward ranging from 10 to 100. To allow for a "safe" starting point for such a dangerous scenario, if two neighboring agents choose zero as their values, neither a reward nor a penalty is obtained. The figure is cropped to highlight the oscillation that occurs with DSA. In domains such as independent path planning of trajectories for UAVs or rovers, in environments where communication channels are unstable, bad assignments could lead to crashes whose costs preclude the use of methods without guarantees of monotonicity.

In addition, monotonicity provides insight as to why coordination might lead to better solution quality. If $k_2 > k_1$, we know that for all assignments x where $x \in X_{k_1 E}$, $x \notin X_{k_2 E}$, there exists an assignment $\tilde{x} \in X_{k_2 E}$ reachable from x such that $\overline{U}(\tilde{x}) > \overline{U}(x)$. This can be seen simply by running MGM-k_2 with initial assignment x.

Example 2. **The Traffic Light Game.** Consider two variables, both of which can take on the values *red* or *green*, with a constraint that takes on utilities as follows:

$$U(red, red) = 0, U(red, green) = U(green, red) = 1, U(green, green) = -1000.$$

Turning this DCOP into a game would require the agent for each variable to take the utility of the single constraint as its local utility. If (red, red) is the initial condition, each agent would choose to alter its value to *green* if given the opportunity to move. If both agents are allowed to alter their value in the same round, we would end up in the adverse state $(green, green)$. When using DSA, there is always a positive probability for any time horizon that $(green, green)$ will be the resulting assignment.

4 Experiments

We considered two domains. The first was a standard graph-coloring scenario, in which a cost of one is incurred if two neighboring agents choose the same color, and no cost is incurred otherwise. Real-world problems involving sensor networks, in which it may be undesirable for neighboring sensors to be observing the same location, are commonly mapped to this type of graph-coloring scenario. The second was a fully randomized DCOP, in which every combination of values on a constraint between two neighboring agents was assigned a random reward chosen uniformly from the set $\{1, \ldots, 10\}$.

In both domains, we used ten randomly generated graphs with 40 variables with three values each, and 120 constraints. We ran: MGM, DSA with $p \in \{.1, .3, .5, .7, .9\}$, MGM-2 with $q \in \{.1, .3, .5, .7, .9\}$ and SCA-2 with all combinations of the above values of p and q (where q is the probability of being an offerer and p is the probability of an uncommited agent acting). Each graph shows an evolution of global solution quality averaged over 100 runs (with random start-states) each for ten examples with selected values of p and q.

We used communication cycles as the metric for our experiments, as is common in the DCOP literature, since it is assumed that communication is the speed bottleneck. However, we note that, as we move from 1-coordinated to 2-coordinated algorithms, the

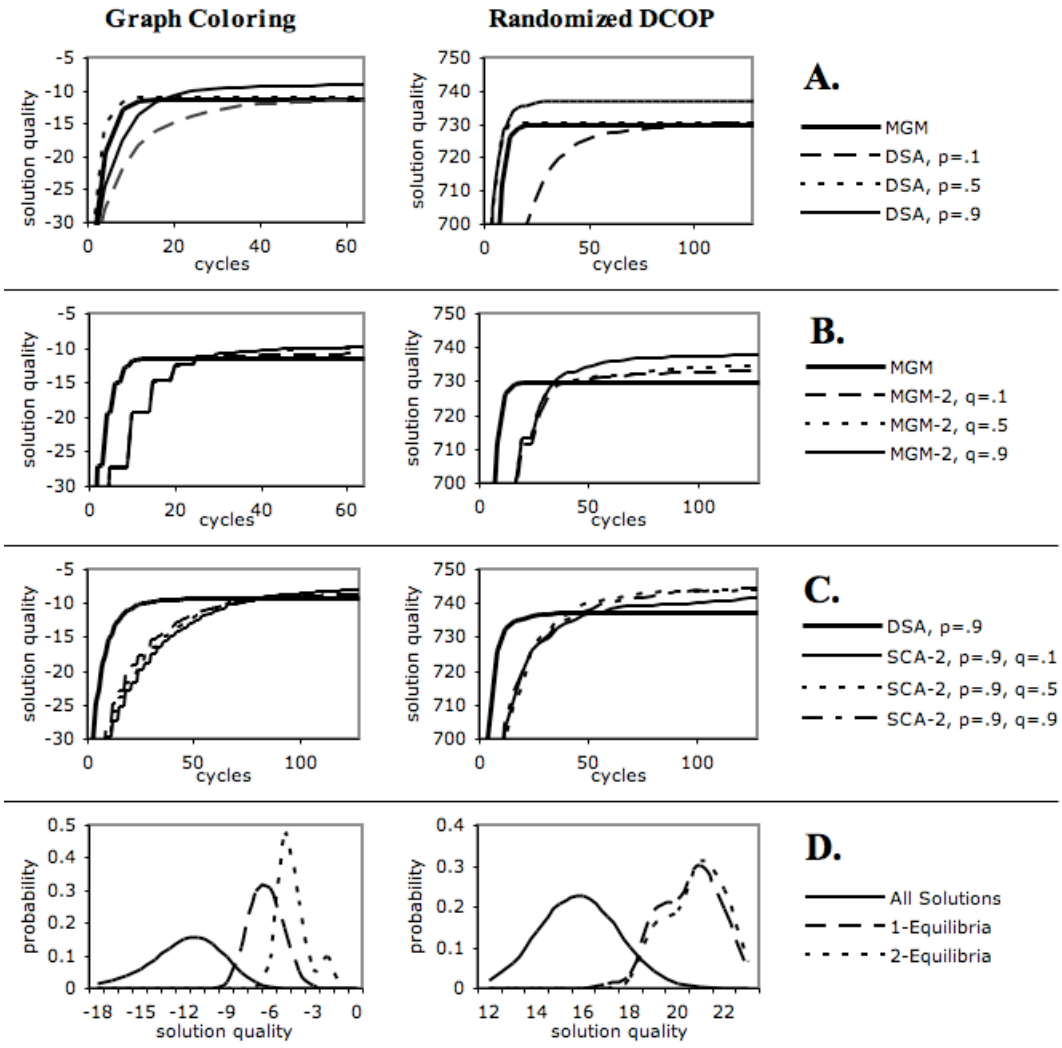

Fig. 3. Experimental results

computational cost each agent i must incur can increase by a factor of as much as $\sum_j |X_j|$ as the agent can now consider the combination of its and all its neighbors' moves. However, in the 2-coordinated algorithms we present, each agent randomly picks a single neighbor j to coordinate with, and so its computation is increased by a factor of only $|X_j|$. Although each run was 256 cycles, the graphs display a cropped view to show the important phenomena.

Figure 3A shows a comparison between MGM and DSA for several values of p. For graph coloring, MGM is dominated, first by DSA with $p = 0.5$, and then by DSA with $p = 0.9$. For the randomized DCOP, MGM is completely dominated by DSA with $p = 0.9$. MGM does better in the high-stakes scenario as all DSA algorithms have a negative solution quality (not shown in the graph) for the first few cycles. This happens because at the beginning of a run, almost every agent will want to move. As the value of p increases, more agents act simultaneously, and thus, many pairs of neighbors are choosing the same value, causing large penalties. Thus, these results show that the nature of the constraint utility function makes a fundamental difference in which algorithm dominates. Results from the high-stakes scenario contrast with [13] and show that DSA is not necessarily the algorithm of choice compared with DBA across all domains.

Figure 3B shows a comparison between MGM and MGM-2, for several values of q. In all domains, MGM-2 eventually reaches a higher solution quality after about thirty cycles, despite the algorithms' initial slowness. The stair-like shape of the MGM-2 curves is due to the fact that agents are changing values only once out of every five cycles, due to the cycles used in communication. Of the three values of q shown in the graphs, MGM-2 rises fastest when $q = 0.5$, but eventually reaches its highest average solution quality when $q = 0.9$, for each of the three domains. We note that, in the high-stakes domain, the solution quality is positive at every cycle, due to the monotonic property of both MGM and MGM-2. Thus, these experiments clearly verify the monotonicity of MGM and MGM-2, and also show that MGM-2 reaches a higher solution quality as expected.

Figure 3C shows a comparison between DSA and SCA-2, for $p = 0.9$ and several values of q. DSA starts out faster, but SCA-2 eventually overtakes it. The result of the effect of q on SCA-2 appears inconclusive. Although SCA-2 with $q = 0.9$ does not achieve a solution quality above zero for the first 65 cycles, it eventually achieves a solution quality comparable to SCA with lower values of q.

Figure 3D shows a probability mass function (PMF) of solution quality for three sets of assignments: the set of all assignments in the DCOP (X), the set of 1-coordinated (Nash) equilibria (X_{1E}), and the set of 2-coordinated equilibria (X_{2E}). Here we considered smaller scenarios with twelve variables, 36 constraints, and three values per variable in order to investigate tractably explorable domains. In both domains, the solution quality of the set of 2-coordinated equilibria (the set of equilibria to which MGM-2 and SCA-2 must converge) is, on average, higher than the set of 1-coordinated equilibria, potentially explaining the higher solution quality of the experimental runs. Even though a higher level of coordination yields better solution quality, the relationship between magnitude of improvement and the difference in solution qualities of the equilibrium sets is not obvious. Trajectories may not be uniformly distributed over the equilibrium sets. Investigating these effects is a ripe area for further investigation.

5 Related Work and Summary

Research in general graphical games has focused on centralized algorithms for finding mixed-strategy Nash equilibria [4, 10]. DCOP games not only guarantee pure-strategy Nash equilibria but also introduce *k*-coordination and hence *k*-coordinated equilibria. In [2], coordination was achieved by forming coalitions represented by a *manager* who made the assignment decisions for all variables within the coalition. These methods require high-volume communication to transfer utility function information and the abdication of authority which is often infeasible or undesired in many distributed decision-making environments. Furthermore, the cost of forming a coalition discourages rapid commitment and detachment from teams. MGM-*k* and SCA-*k* allow for coordination while maintaining the underlying distributed decision-making process and allowing dynamic teaming in each round.

A fundamental novelty of our approach is our analysis of distributed *k*-coordination algorithms as well as *k*-coordinated equilibria. The key contributions of this paper include: (i) an introduction of *DCOP games* for analysis of DCOP algorithms, (ii) development of *k*-coordinated DCOP algorithms, (iii) identification of a mapping between finite games and coding theory leading to *a priori* bounds on cardinality of equilibria sets of *k*-coordinated algorithms, (iv) improvement on the tightness of current bounds, (v) proof of monotonicity of the MGM-*k* class of algorithms and (vi) an investigation of the equilibria sets of algorithms of differing degrees of coordination.

We provided key experimental results, verifying our conclusions about monotonicity and equilibria bounds. This paper is a significant extension of the authors' previous work in DCOP games [7], in which *k*-coordinated algorithms and equilibria were introduced. Our results comparing 1-coordinated and 2-coordinated algorithms illustrate the need to develop efficient *k*-coordination algorithms for higher *k* in the future.

6 Acknowledgment

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA), through the Department of the Interior, NBC, Acquisition Services Division, under Contract No. NBCHD030010.

References

1. S. Fitzpatrick and L. Meertens. Distributed coordination through anarchic optimization. In V. Lesser, C. L. Ortiz Jr., and M. Tambe, editors, *Distributed Sensor Networks: A Multiagent Perspective*, pages 257–295. Kluwer Academic Publishers, 2003.
2. K. Hirayama and J. Toyoda. Forming coalitions for breaking deadlocks. In *Proc. ICMAS*, pages 155–162, 1995.
3. J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79:2554–8, 1982.
4. M. Kearns, M. Littman, and S. Singh. Graphical models for game theory. In *Proc. UAI*, pages 253–260, 2001.
5. S. Ling and C. Xing. *Coding theory: A first course*. Cambridge University Press, 2004.

6. F. J. MacWilliams and N. J. A. Sloane. *The theory of error-correcting codes*. North-Holland, 1977.
7. R. T. Maheswaran, J. P. Pearce, and M. Tambe. Distributed algorithms for DCOP: A graphical-game-based approach. In *PDCS 2004*, San Francisco, CA, September 2004.
8. R. T. Maheswaran, M. Tambe, E. Bowring, J. P. Pearce, and P. Varakantham. Taking DCOP to the real world: efficient complete solutions for distributed multi-event scheduling. In *AAMAS 2004*, New York, NY, July 2004.
9. P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. An asynchronous complete method for distributed constraint optimization. In *Proceedings of the Second International Conference on Autonomous Agents and Multi-Agent Systems*, Sydney, Australia 2003.
10. D. Vickrey and D. Koller. Multi-agent algorithms for solving graphical games. In *Proc. AAAI*, pages 345–351, 2002.
11. M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):673–685, 1998.
12. M. Yokoo and K. Hirayama. Distributed breakout algorithm for solving distributed constraint satisfaction and optimization problems. In *Proc. ICMAS*, pages 401–408, 1996.
13. W. Zhang, Z. Xing, G. Wang, and L. Wittenburg. An analysis and application of distributed constraint satisfaction and optimization algorithms in sensor networks. In *AAMAS 2003*, pages 185–192, Melbourne, Australia, July 2003.

Appendix A: Algorithms

Algorithm 1 MGM (allNeighbors, currentValue)

```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: SendGainMessage(allNeighbors,gain)
5: neighborGains = ReceiveGainMessages(allNeighbors)
6: if gain > max(neighborGains) then
7:     currentValue = newValue
8: end if
```

Algorithm 2 DSA (allNeighbors, currentValue)

```
1: SendValueMessage(allNeighbors, currentValue)
2: currentContext = GetValueMessages(allNeighbors)
3: [gain,newValue] = BestUnilateralGain(currentContext)
4: if  Random(0,1) < threshold then
5:     currentValue = newValue
6: end if
```

Algorithm 3 MGM-2 (allNeighbors, currentValue)

```
SendValueMessage(allNeighbors, currentValue)
currentContext = GetValueMessages(allNeighbors); committed = no
if  Random(0,1) < offererThreshold then
    committed = yes; partner = RandomNeighbor(allNeighbors)
    SendOfferMessage(partner,allCoordinatedMoves(partner))
end if
[gain,newValue] = BestUnilateralGain(currentContext)
offers = ReceiveOffers(allNeighbors); offerReplySet = ∪ {offers.neighbor}
if committed = no then
    bestOffer = FindBestOffer(offers)
    if bestOffer.gain > gain then
        offerReplySet = offerReplySet \{ bestOffer.neighbor}
        committed = yes; partner = bestOffer.neighbor
        newValue = bestOffer.myNewValue; gain = bestOffer.gain
        SendOfferReplyMessage(partner, commit, bestOffer.partnerNewValue, gain)
    end if
    for all neighbor ∈ offerReplySet do
        SendOfferReplyMessage(neighbor, noCommit)
    end for
end if
if committed = yes then
    reply = ReceiveOfferReplyMessage(partner)
    if reply = commit then
        newValue = reply.myNewValue; gain = reply.gain
    else
        committed = no
    end if
end if
SendGainMessage(allNeighbors,gain)
neighborGains = ReceiveGainMessages(allNeighbors); changeValue=no
if committed=yes then
    if gain > max(neighborGains) then
        SendConfirmMessage(partner, go)
    else
        SendConfirmMessege(partner, noGo)
    end if
    confirmed = ReceiveConfirmMessage(partner)
    if confirmed=yes then
        changeValue=yes
    end if
else
    if gain > max(neighborGains) then
        changeValue=yes
    end if
end if
if changeValue=yes then
    currentValue = newValue
end if
```